Don’t @ Me: Hunting Twitter Bots at Scale

Jordan Wright, Principal R&D Engineer, and Olabode Anise, Data Scientist, Duo Security

Introduction

Key Findings

Current Landscape of Twitter Bot Research

Building the Dataset
- Finding Twitter Accounts
  - How Twitter Assigns User IDs
  - 32-Bit User IDs
  - 33-Bit User IDs
  - Snowflake IDs
    - Using the Tweet Stream
- Enriching the Dataset
  - Gathering Tweets
    - Fetching Replies
  - Mapping the Social Network
  - Rate Limit Summary
- System Architecture

Anatomy of a Twitter Bot
- Identifying Heuristics
  - Account Attributes
  - Content
  - Content Metadata

Classifying Bot Accounts at Scale
- Dataset
- Why Use Machine Learning
- Classification via Account Metadata
  - Exploratory Data Analysis
  - Model Training and Evaluation
  - Results on Unknown Accounts
- Classification via Tweet Behavior, Tweet Content and Account
  - Feature Selection

Results on Unknown Data
Limitations, Considerations and Other Discussion

Crawling Botnets
Case Study: Cryptocurrency Scam Botnet
Initial Discovery
Evolution of the Botnet
Mapping the Network
Mapping Followers
Mapping Amplification Bots

Approaches That Didn't Work
Snowflake ID Backtracking
Finding Who Favorited a Tweet

Next Steps

References

Appendix
Feature List for Machine Learning Algorithms
B. Cross-Validation Results

Introduction

Automated Twitter accounts have been making headlines for their effectiveness at spreading spam and malware as well as influencing online discussion and sentiment. There have been a variety of methods published in previous work showing either how to create a Twitter dataset [4], [16] or techniques for finding potential bots from a previously shared or gathered dataset [20], [32]. However, we have yet to find a work that describes the entire process in detail, providing tools and techniques to other researchers interested in finding automated Twitter accounts.

This paper details the techniques and tools we created to both build a large dataset containing millions of public Twitter profiles and content, as well as to analyze the dataset looking for automated accounts. By applying a methodical data science approach to analyzing our dataset, we were able to build a classifier that effectively finds bots at a large scale. In this work, we specifically look for automated accounts, **not necessarily malicious automated accounts**. Distinguishing benign automation from malicious automation is a topic for future work.

We then demonstrate how to pivot off discovered bots to find entire communities of connected bots, or "botnets." This analysis is given through a case study detailing a large organized cryptocurrency spam botnet.

We have open sourced our data collection system used to gather account, tweet and social network data to enable the community of security researchers to build on our work. This system allows researchers to quickly build a large Twitter dataset for use in future work.

Finally, we aim to provide a narrative to our research, explaining why we chose various approaches. We then include a section at the end of the paper that highlights different techniques we tried that didn't yield the expected results for the purposes of providing transparent research.

Key Findings

This paper presents the following key findings:

- Using knowledge of how Twitter generates user IDs, we gathered a dataset of 88 million public Twitter profiles consisting of standard account information represented in the Twitter API, such as screen name, tweet count, followers/following counts, avatar and description.
- As API limits allow, this dataset was enriched with both the tweets posted by accounts, as well as with targeted social network information (follower/following) information.
- Practical data science techniques can be applied to create a classifier that is effective at finding automated Twitter accounts, also known as "bots."
- A case study detailing a large botnet of at least 15,000 bots spreading a cryptocurrency scam. By monitoring the botnet over time, we discover ways the bots evolve to evade detection.
- Our cryptobot scam case study demonstrates that, after finding initial bots using the tools and techniques described in this paper, a thread can be followed that can result in the discovery and unraveling of an entire botnet. For this botnet, we use targeted social network analysis to reveal a unique three-tiered hierarchical structure.

Current Landscape of Twitter Bot Research

The dynamics of how users communicate on Twitter make it a fascinating area of both social and security-related study. In particular, the topic of building datasets based on content posted to Twitter and identifying automated accounts within a dataset are two areas we explore in this paper.

[4] and [16] use the Twitter API to enumerate large datasets of users and their social networks to characterize attributes about the users and trends in the growth of Twitter's user base.

[5] and [6] use information about how user IDs are generated to gather a sample dataset used to identify two large botnets. As noted in the “Finding Twitter Accounts” section below, our work follows a similar methodology when building the dataset, though we extend data collection to the previously unstudied 33-bit ID space. In addition to this, we examine the challenges and workarounds when studying user IDs generated with the 64-bit Snowflake format.

Studies such as [5], [6], [18], [21], [32], [36] and [37] use attributes from gathered datasets to build classifiers which are used to identify automated accounts. Our work uses a similar approach to build a classifier that is used to find automated accounts within our gathered dataset.

To discover organized botnets, studies such as [5] and [6] build classifiers with specific features to find related bots. Due to limitations imposed by Twitter’s API, it’s infeasible to study organized botnets across a large dataset using social network information. As a case study, our paper analyzes a large cryptocurrency scam botnet, showing how targeted social network information gathering can reveal the structure of the botnet.

Our work provides an end-to-end look at both the techniques, tools and challenges of gathering large Twitter datasets as well as showing the process of applying a practical data science approach to discover automated bots.

Finally, this work introduces our open-source data collection system [35], which can be used to gather public datasets suitable for use in further study by the larger research community.

Building the Dataset

The first part of our research involved creating a system that efficiently gathers large amounts of public Twitter data for analysis. At a high level, this process used the Twitter API [1] to gather an initial dataset of public profile information as shown in the “Finding Twitter Accounts” section. We then enrich the dataset by gathering the tweets associated with accounts. As needed, it’s possible to then fetch the social network connections for each account, such as the following/followers information.

When creating a data collection system, we intentionally constrained ourselves to the rate limits imposed by the Twitter API so that the system could be used by other researchers without modification. This means that the data collected from this system used a single application with a single authenticated user.

In the following sections, we describe techniques we used to maximize the efficiency of our data collection while conforming to the rate limits imposed by the API.

Finding Twitter Accounts

Fetching public Twitter profiles is the first step toward building our dataset. The Twitter API represents high-level profile information as user objects. User objects contain fields including the profile's screen name, tweet count, followers/following counts, avatar and description [2].

Our goal is to find accounts as efficiently as possible. That is, to maximize the rate in which new accounts are discovered while still conforming to the imposed API limits. In addition to this, where possible, we aim to create a uniform, random sample of accounts that is representative of the larger Twitter ecosystem.

Enumerating user accounts first relies on an understanding of how Twitter generates user IDs.

How Twitter Assigns User IDs

Twitter first assigned user IDs as sequential 32-bit unsigned integers from 1 to 2^32 [4]. This range has been used in previous studies such as [5] and [6] when enumerating user accounts. In the "33-Bit User IDs" section below, we show that our data gathering discovered that this sequential ID assignment continued into the 33-bit space.

In 2013, Twitter announced [7] that account IDs would transition to a new 64-bit format generated by a system called Snowflake. Initially introduced in 2010 [8], Snowflake had previously been generating IDs for tweets, but was being transitioned to generating IDs for most (if not all) Twitter objects.

Snowflake IDs have the following format:

- **41 bits:** The timestamp - number of milliseconds since a custom epoch [27]
- **10 bits:** Node number - a unique number to identify the host that generated the ID
- **12 bits:** Sequence number - a rolling sequence number used by each host

In the blog post [7], Twitter scheduled the switch to Snowflake IDs for October 2013. The earliest 64-bit ID found in our dataset was created in early 2016, suggesting a later transition date. This also suggests that, while the exact cause of extending account IDs into the 33-bit unsigned integer space is unknown, it's possible that the 33-bit ID generation served as a transition period both for Twitter to move to the new ID format, as well as for client libraries to adapt to the new, longer ID format which was known to cause inconsistencies for certain languages [31].

Using this knowledge of account ID generation, we gathered the account dataset using two separate methods: account ID enumeration and hooking into the Twitter sample tweet stream.

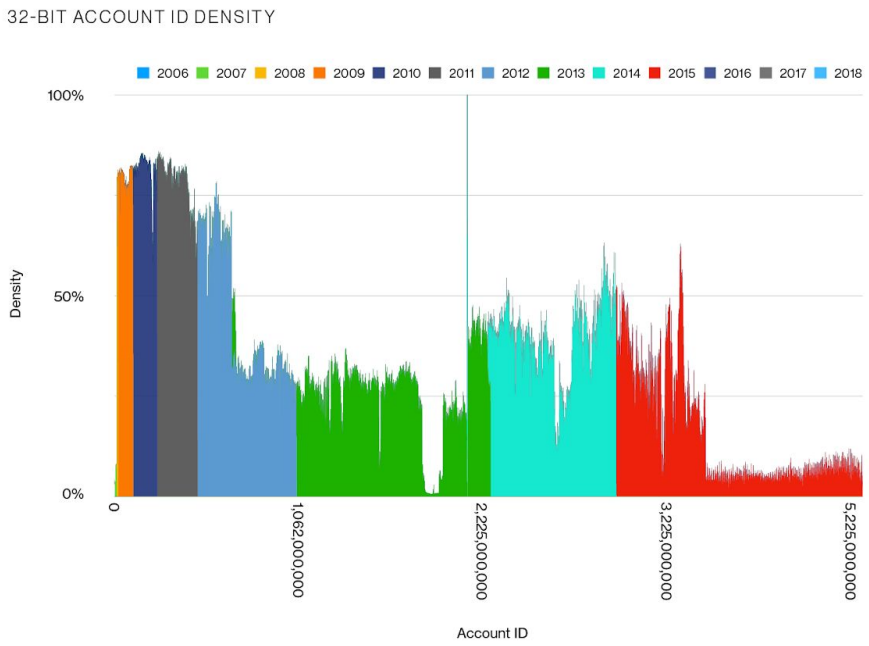
32-Bit User IDs

Twitter offers a variety of API endpoints to fetch account information. One of the most efficient endpoints is `users/lookup` [3], which offers the ability to submit 100 possible user IDs or screen names per request and receive the fully populated user objects for existing accounts. This API endpoint has a rate limit in place which, at the time of writing, allowed an authenticated user to submit 900 requests over a 15-minute window. This means that by finding a way to reliably generate candidate user IDs, this endpoint can be used to perform 8.6 million ID lookups per day.

As mentioned in the previous section, user IDs were first generated as sequential unsigned 32-bit integers. This makes it easy to create a uniform, random sampling of the entire ID space by generating a percentage of candidate IDs and submitting them against the `users/lookup` method.

Having a uniform sample set across the entire 2^32 bit ID space provides a large corpus of account data that can be used to discover groups of bots created around the same time. For example, previous work from Echeverria [5] and [6] was successful at finding large botnets sourced from a 1 percent random sampling of this available ID space.

During the course of our research, we enumerated a random 5 percent sampling over the available ID space. Similar to the work from [6] the following chart shows the density of accounts in this available space, using buckets of 1 million account IDs and colored by the year the account was created.



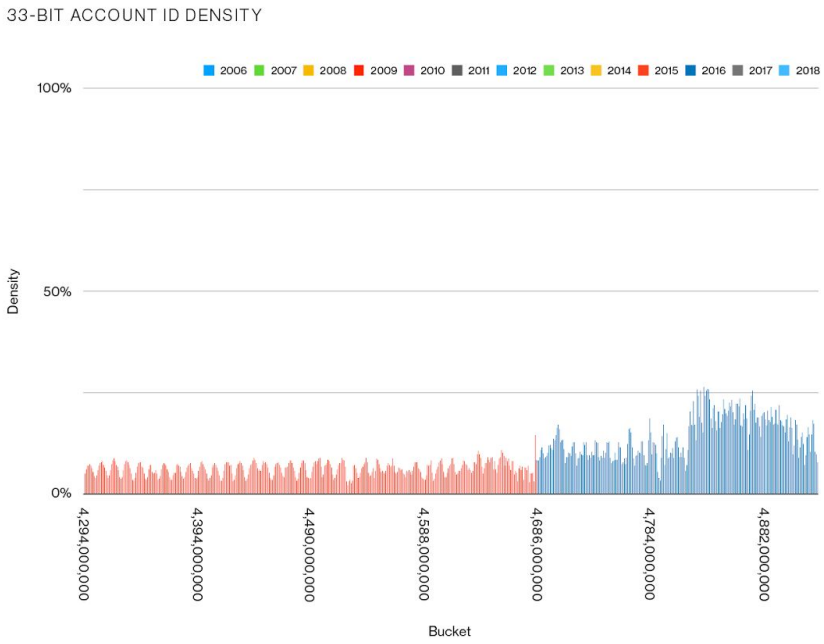
At an average account density of 32 percent, this yielded a uniform random sample of 66 million accounts.

33-Bit User IDs

Previous work had only discussed account IDs in the 32-bit unsigned integer space; however, during our account gathering process, we also discovered that the use of sequential IDs continued throughout 2016 with IDs extending into the sequential 33-bit unsigned integer space.

We extended our 5 percent account sampling to include the space between 4*10^6 and 5*10^6, yielding roughly 3 million new accounts largely created between 2015 and 2016. During our data gathering, we did not find any accounts with an ID greater than 5*10^6.

The following graph shows the ID space density we recorded in this ID range:



Snowflake IDs

The introduction of the Snowflake ID format makes it difficult to enumerate accounts sequentially, since there are multiple different pieces of information that would need to be guessed. In the “Approaches that Didn’t Work” section, we describe the results of our attempts to enumerate Snowflake IDs by analyzing our existing ID corpus.

Since enumeration isn’t feasible, we have to take a different approach by letting the user information come to us by hooking into Twitter’s tweet stream.

Using the Tweet Stream

Twitter offers multiple different streams, which are API endpoints that allow clients to connect and receive real-time tweet objects filtered and sampled using various parameters. While most of the streaming endpoints are reserved for paid enterprise access, two of the streaming endpoints, `statuses/sample` and `statuses/filter`, are available to all applications.

For our research, we connected to the `statuses/sample` endpoint, which provides a “small random sample of all public statuses” [9]. The tweet objects returned in this stream contain the full user object of the account that created the tweet.

While in our case we chose to use the `statuses/sample` endpoint, the `statuses/filter` endpoint can be used to support targeted research around specific events through hashtags or keywords. This is also useful, for example, to find tweets containing links by filtering on the term “http.”

When building a dataset, it's important to remember what biases are being included. As an example, we note that by retrieving accounts through the Twitter stream, this data is biased towards users who have tweeted very recently. When building our classifier in the "Classifying Bots at Scale" section below, we do not use how recently an account has tweeted as an attribute of bot behavior. We more comprehensively address potential limitations in the dataset in the "Limitations, Considerations and Other Discussion" section later in the paper.

During the course of our research, we gathered 19 million accounts via the streaming API, averaging approximately 940,000 accounts per day.

Enriching the Dataset

After gathering the high-level account information, the next step is to enrich the dataset with both the content (tweets) and the social network information (how accounts are connected).

While it's possible to predict with a certain degree of confidence whether or not an account is a bot using just the account information provided in the user object, there are other attributes that are only available by analyzing the tweet content. Additionally, gathering the social network information helps identify directed connections between bot accounts, as well as identifying potential communities, also known as "botnets."

Gathering Tweets

The first step in the data enrichment process is gathering the tweets for an account. The `statuses/user_timeline` endpoint [10] can be used to fetch the latest 3,200 tweets from an account's timeline.

However, this endpoint only returns 200 tweets per request, and is rate limited to 1,500 requests per rate limit window. Since this method only accepts a single screen name or user ID per request, this causes the tweet collection process to be much slower than the account collection with just 144,000 lookups per day.

To help make tweet collection as efficient as possible, we applied the following rules when fetching tweets for the accounts in our dataset:

- Only fetch tweets for users with more than 10 tweets published
- Only fetch tweets for users who don't have their account set to "protected"
- Only fetch tweets for users who declare English as their language

In addition to these, we only fetched the first 200 tweets for each user. We determined that, for our research, the latest 200 tweets would be a large enough sample size for determining if the account was automated, as well as what type of content the account published.

We can also use the tweets to help discover new user accounts. Tweet objects contain the account IDs for any mentioned users. Additionally, if the tweet is a reply to another tweet, the original account ID is also included in the `in_reply_to_user_id` field. These extracted IDs can be sent back into the `users/lookup` process to gather the full accounts of users found in tweets.

During the course of our research, we gathered tweets for 3.1 million accounts, resulting in a dataset of 576 million tweets.

Fetching Replies

If the fetched tweet is a retweet, then the original tweet is included in the response object. However, if the tweet is a reply, the response only specifies the original tweet ID. Some characteristics of bot accounts apply to the relationship between the published reply and the original tweet, such as how quickly the reply was created. In order to analyze the relationship between the reply and the original tweet, we needed to fetch the original tweet.

To do this, we can use the `statuses/lookup` [11] endpoint. This method has the same rate limits as the `users/lookup` method used earlier in the account ID enumeration process.

Mapping the Social Network

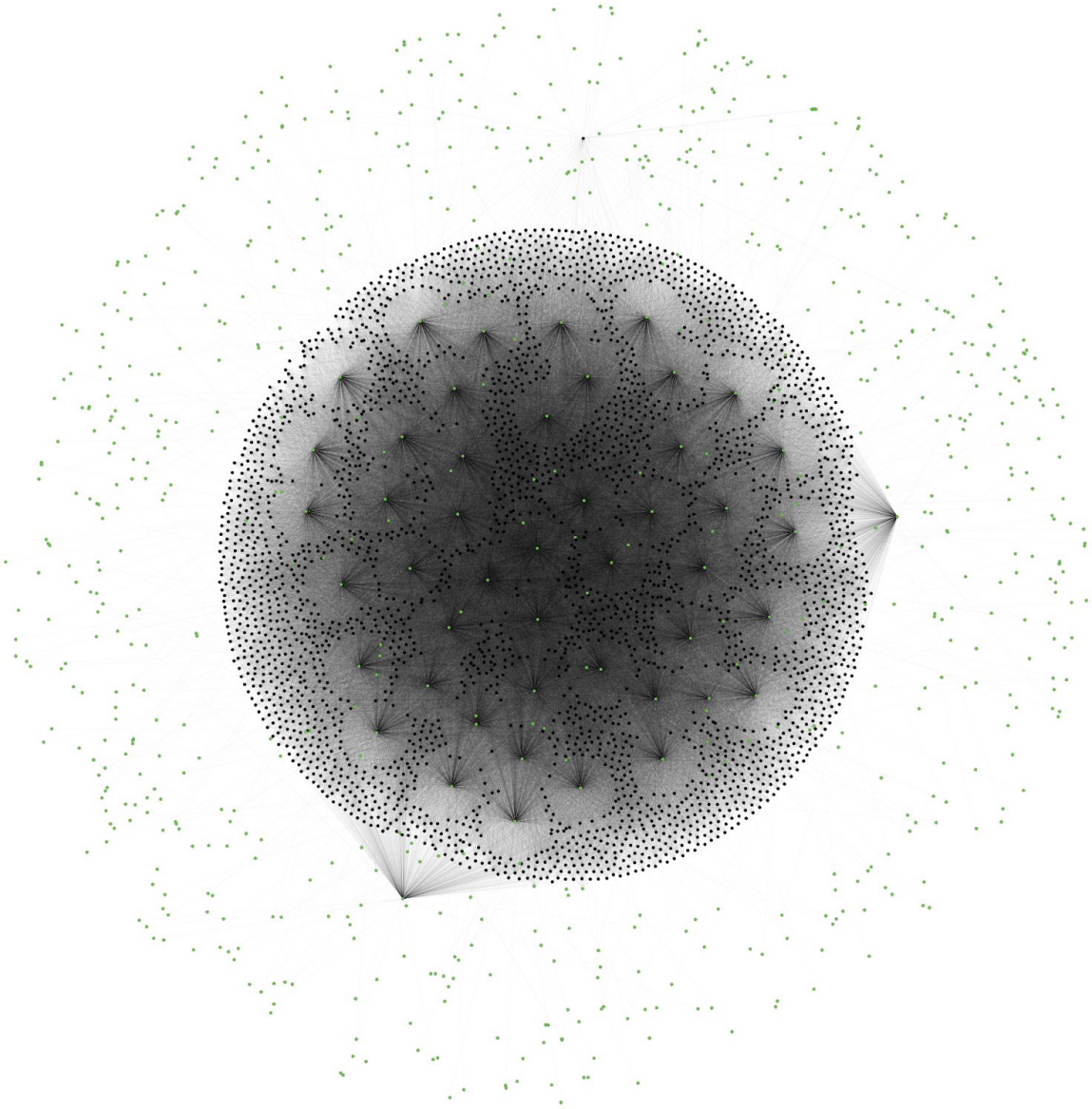
One of the goals of our research was to explore how bot accounts are controlled and how they connect to each other. For example, if accounts tweet identical content at roughly the same time, then we can infer with a degree of confidence that the accounts are controlled by the same entity. However, another way to group accounts is by analyzing their social network connections.

Twitter tracks directional connections in the form of “follow”s. The `friends/ids` [12] and `followers/ids` [13] API endpoints can be used to retrieve the account IDs of who the targeted account is following, or who is following the account. These endpoints return IDs in groups of 5,000, which can then be used with the `users/lookup` endpoint to retrieve the full account information. This process can then be repeated for each new connection, resulting in a network of accounts that are considered an *n-degree* connection to the original account.

Once these connections are retrieved, a directional graph can be built to further analyze and explore using specialized graph analysis tools like Gephi [14] or Neo4j [15]. As part of this research, we’ve open sourced a script, `crawl_network.py`, [35] that creates the social graph to a configurable degree for a given account and exports the results in GEXF format.

The graph below demonstrates this output, showing how a 1-degree crawl of the social network for a known “fake follower” (a bot account that exists to follow other accounts in order to artificially inflate those accounts’ popularity) can discover a large network of thousands of

other fake followers (black nodes) highly connected to potentially legitimate user accounts (green nodes).



During our research, both endpoints were rate limited to only allow 15 requests per 15-minute rate limit window. Previous research [16] attempted to map the entire Twitter social graph by finding systems that had been previously whitelisted from Twitter's API rate limits. For our research, we explicitly wanted to build a system that gathers data conforming to the normal API rate limits. This means that the existing rate limits on these endpoints make gathering social network connections at a large scale impractical due to the time requirements.

However, gathering social network information can still be very useful when applied in a targeted fashion. For example, once a bot is identified, the bot's social network information can be gathered to find similar connected accounts, resulting in a network that could be a potential botnet. An application of this targeted network analysis to discover an organized botnet can be found in the "Case Study: Cryptocurrency Spam Botnet" section below.

Targeted social network gathering can also be useful in discovering bots designed to artificially inflate follower counts. These bots, called "fake followers," are used to increase the number of followers for an account, making the account seem more popular than it actually is. The *New York Times* published an article [17] that demonstrated patterns of bot behavior when viewing the account creation dates for followers over time.

Rate Limit Summary

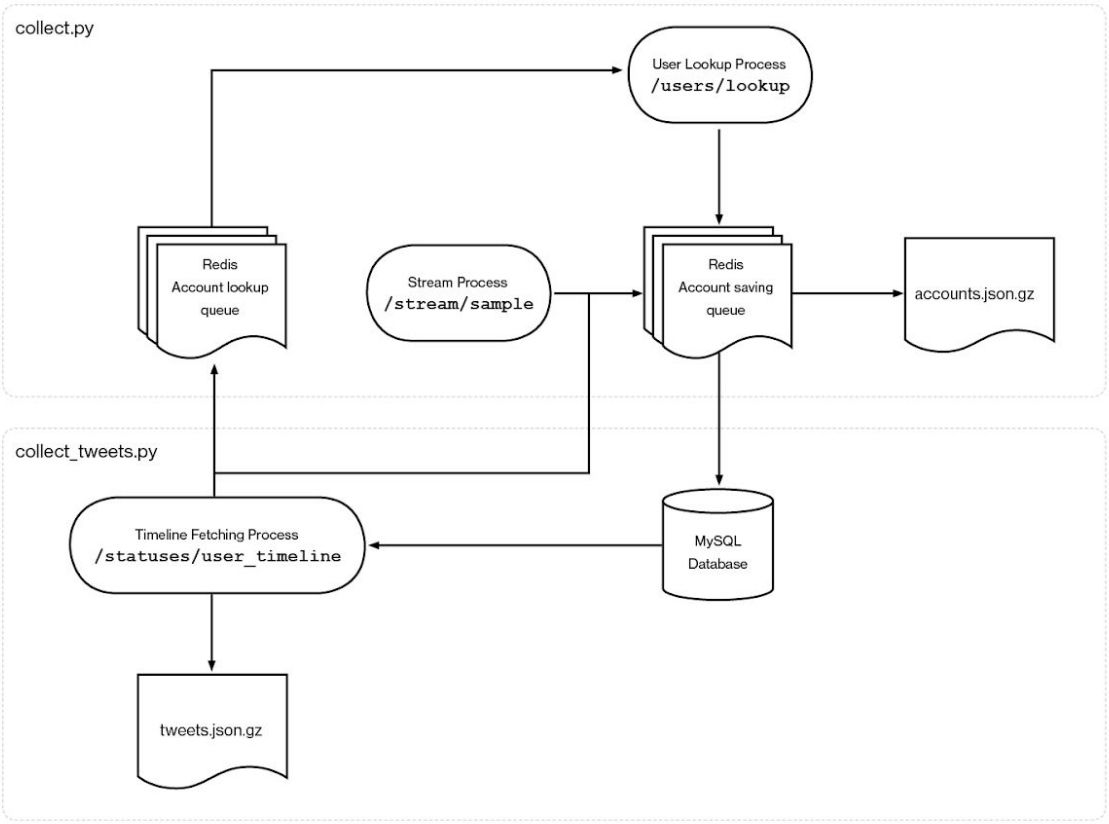
The previous sections explain the various measures we took to gather account information as effectively as possible, however, the output of the system is still largely determined by the rate limits in place on the APIs.

The following chart summarizes the rate limits in place for the various API endpoints used during our data collection:

System Architecture

To enable security and academic researchers to reproduce our results or perform their own research on large-scale public Twitter datasets, we've open sourced our data gathering system [35], which uses the techniques detailed above to efficiently gather accounts and tweets.

The following architecture diagram shows the components of the data gathering system:



This system is divided into two separate processes: an account collection process (`collect.py`), and a tweet collection process (`collect_tweets.py`). The final information is saved in compressed JSON files in NDJSON format. High-level account metadata is also saved in a database, which can be used for summarized analytics.

Anatomy of a Twitter Bot

After gathering the initial dataset, we started the process of identifying bot accounts. To do this, we asked the question “what attributes would make an account look suspicious?” We then took these characteristics and manually verified a small number of accounts to see if our decisions proved effective in finding bots. Then, in the “Classifying Bot Accounts at Scale” section below, we construct a classifier to more effectively identify bots across our dataset.

Identifying Heuristics

We divided the attributes of an account into three categories:

- **Account Attributes** - These are attributes in the user object, including the number of tweets, likes, following/follower count, screen name and more

- **Content** - The text of the created tweets
- **Content Metadata** - The metadata for created tweets, including time-based and location-based information

A full list of the attributes we included can be found in the following table:

Heuristic	**Category**
Number of digits at the end or beginning of the screen name	Account attributes
Entropy of the screen name	Account attributes
Ratio of followers/following	Account attributes
Number of likes relative to account age	Account attributes
Number of tweets relative to account age	Account attributes
Average number of users mentioned in a tweet	Content
Average number of unique accounts being retweeted	Content
Number of unique sources	Content
Number of tweets with the same content per day	Content
Percentage of tweets with URLs	Content
Ratio of tweets with photos vs. just text	Content
Average number of hashtags in tweet	Content
Number of unique languages detected in tweets	Content metadata
Number of tweets per day, relative to account age	Content metadata
Average number of tweets per day	Content metadata
Average time between tweets	Content metadata
Average hours tweeted per day	Content metadata
Average “distance” of account age in retweets/replies	Content metadata
Average distance between geolocation-enabled tweets	Content metadata

It’s important to note that these aren’t the only attributes we could have considered. There have been studies such as [36] and [37] that used these and other attributes when studying

automated Twitter accounts. The attributes that we studied were areas we felt would lead to effective bot detection, but we encourage researchers to explore new attributes which may be even more effective.

The next sections briefly explore the various attributes and why we considered them useful when finding bots.

Account Attributes

Account attributes, such as the number of tweets or likes, can be useful for identifying a bot when compared to the length of time the account has been active, as well as when compared to other attributes. Some sources such as [34] use account information as a primary source when identifying fake followers.

Additionally, we observed cases where accounts had no activity other than liking many tweets in a short period of time, raising the possibility that the account may be an amplification bot designed to artificially inflate the popularity of tweets.

We also analyzed attributes of the screen name, such as the amount of entropy or the number of digits at the beginning or end. This helps find bots that have algorithmically-generated screen names, which is often required when generating large amounts of accounts.

Content

Once tweets for an account are gathered, they can be analyzed to identify bots that aim to spread spam and malicious links to unsuspecting users. While there have been studies such as [18] that analyze the language structure and sentiment to discover bots that influence online discussion, our analysis focused on attributes that were more straightforward to calculate.

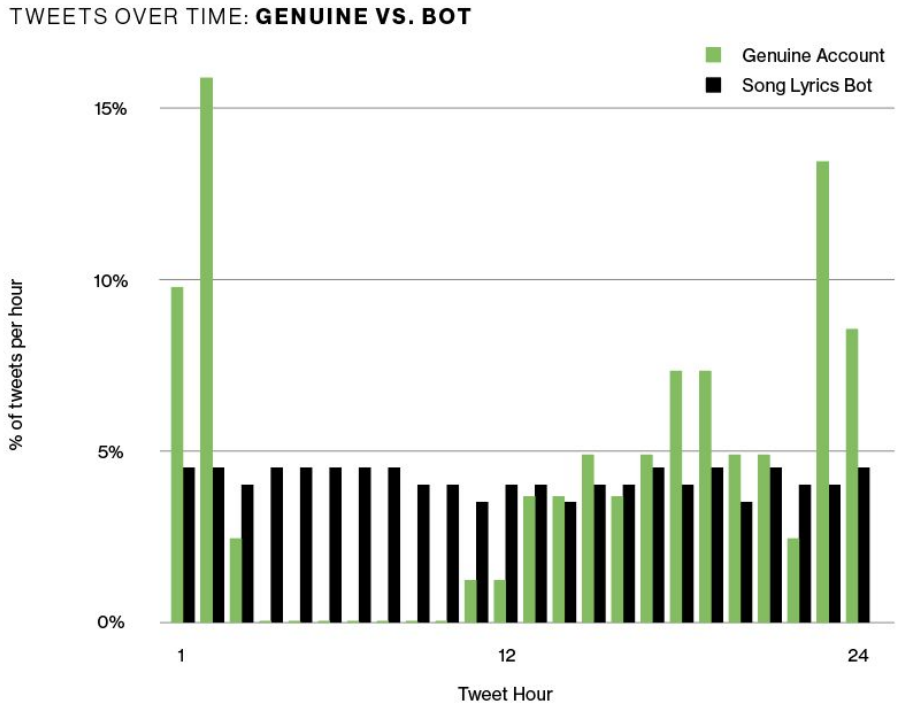
Many of the attributes we analyzed were focused on the number or inclusion of different elements within a tweet. For example, we averaged the number of tweets each account created that includes a URL. This may be indicative of a user who actively shares links, tweets that were created by a third-party application, or potentially a bot sharing malicious links.

Content Metadata

One of the most important types of attributes to consider when analyzing an account for bot activity is the metadata around posted content. This metadata can identify suspicious patterns when aggregated over all of the account's tweets.

A great example of this is **time**. For example, the average Twitter user will likely only tweet during certain hours of the day, whereas bots are able to tweet throughout the entire day. Averaging the number of hours per day an account is active may help identify automated accounts.

The following graph illustrates the differences when comparing the distribution of tweets over time for both an individual and a bot:



This same analysis can be performed on other tweet relationships, such as how quickly consecutive tweets are posted, how quickly replies are generated, or how quickly a tweet is retweeted.

Another type of metadata used for identifying bots is **geolocation**. While adding geolocation to tweets is optional, there have been cases of bots doing this, likely an attempt to evade detection. The “Star Wars” botnet in [5] was initially discovered by examining accounts with tweets claiming to originate from a uniformly distributed rectangular area, including uninhabited areas.

In addition to the location from where the tweets were posted, another pattern to analyze is the **average distance between tweets**. If bot creators always update their geolocation for every tweet, the average distance between these tweets may be much higher than a normal user.

Classifying Bot Accounts at Scale

Dataset

The dataset we used in our project comprised accounts labeled as genuine (labeled genuine accounts) or social spam bots by Cresci et. al in 2016 along with their tweets [20]. In addition, we included the account metadata and tweet data associated with accounts that were involved in a cryptocurrency giveaway scam (labeled crypto-giveaway bots) discussed in the "Crawling Botnets" section below.

Last, we have the 42 million unlabeled accounts along with their tweets that were compiled by using Twitter's APIs. Across each of the groups, there are 15,636 accounts and 6,389,009 tweets. Below you can find a breakdown of each of the different groups.

Dataset	Number of Accounts	Number of Tweets	Composition
Genuine accounts	3,474	2,839,361	Labeled genuine user accounts
Social spam bots	4,912	3,457,133	Labeled bot accounts
Cryptocurrency scam bots	7,250	92,515	Labeled bot accounts
Unknown Dataset	4,2010,524	303,483,004	Entirely unknown

Goals and Objectives

In the following sections, we will:

1. Experiment with a few classification algorithms and show how they perform on a labeled dataset without any hyperparameter tuning
2. Illustrate how performance changes when training on one class of bots and predicting on another when using the best model
3. Determine which combination of features are most important
4. Gain an understanding of the accounts that our model believes to be most likely to be a bot through the use of the millions of accounts that we compiled

Why Use Machine Learning

When we used the attributes listed above to create heuristics, we had to manually examine outliers to determine if the attribute would be effective at finding bots. However, this process doesn't scale to the millions of accounts in our unknown dataset or even the thousands of

accounts in our labeled datasets. Instead, we leveraged a few machine learning algorithms, using a subset of the attributes listed in the table above to build features which are used to train a classification algorithm to determine whether an account is a bot. The final list of features is in Section A of the Appendix.

Fitting a model that can be used for classification with our attributes has the benefit of viewing an account through the lens of a combination of all our features, not each individually. Some of our features are effective at identifying bots when combined with other features, but would have a very high false-positive rate if used in isolation.

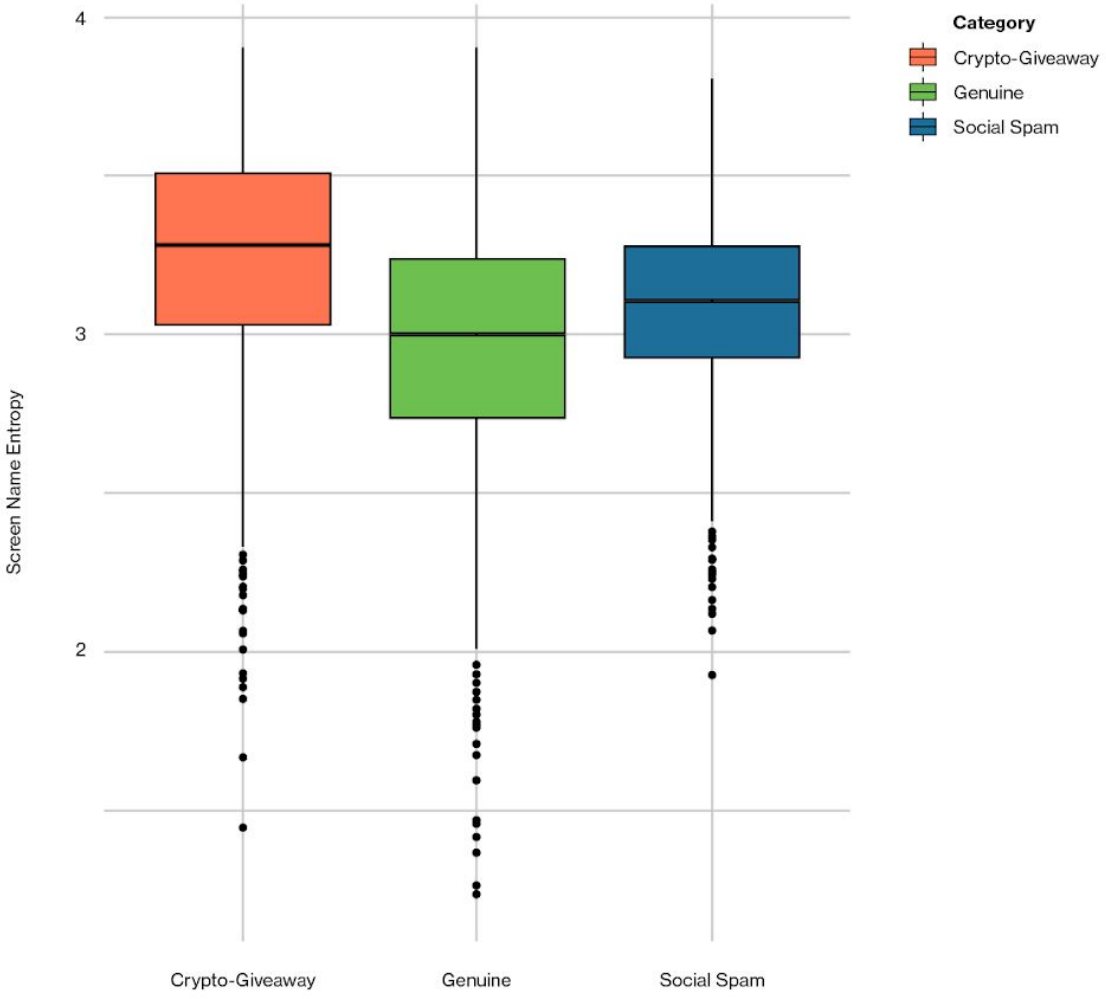
For example, the percentage of tweets with links had a mixture of both automated accounts and legitimate news sources or content creators. These models can optimize how much emphasis, or weight, is put on each feature to help increase the accuracy of bot detection. There is a substantial amount of literature showing that approaches relying on labeled (supervised) [19] and [20] and unlabeled datasets (unsupervised) [21] have proven to be successful in classifying bots and/or automation on Twitter.

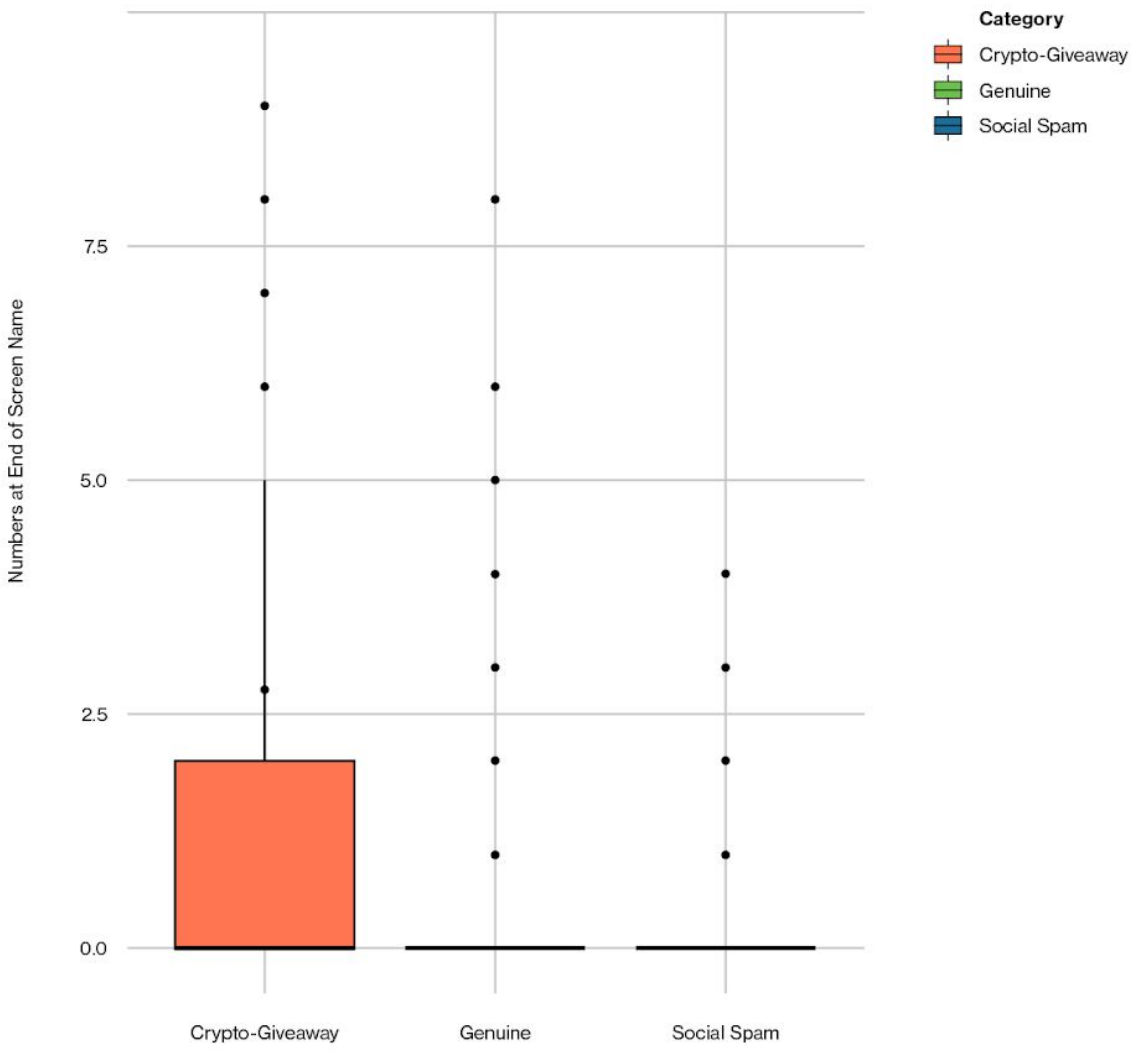
Classification via Account Metadata

Exploratory Data Analysis

Prior to investigating which algorithm would perform the best on our data, we explored each of the predictors we detailed above and how they differed among each of the groups. In terms of the predictors that measure activity (e.g., number of tweets per day), the genuine accounts had the highest averages by far. Those genuine accounts averaged 3.78 favorites per day and 11.3 tweets per day.

For both sets of bots, their averages were at least a degree of magnitude lower. In terms of attributes associated with screen names, the bot accounts showed some distinct differences. On average, both sets of bot accounts had screen names with higher Shannon entropy; however, only the crypto-giveaway bots had a distinguishable number of characters at the end of the screen name.


Category
Crypto-Giveaway
Genuine
Social Spam
Screen Name Entropy
4
3
2
Crypto-Giveaway
Genuine
Social Spam



Model Training and Evaluation

Using the following features:

- `favorite_rate` - ratio of favorites to account age
- `ratio_followers_friends` - the number of followers divided by the number of friends or accounts the user is following
- `is_geo_enabled` - binary feature of whether or not geo-coordinates are accessible
- tweet_rate - ratio of number of tweets to account age
- `is_default_profile` - binary feature of whether or not the user has altered the default profile
- `screen_name_entropy` - Shannon entropy of a user's screen name

- `numbers_at_end_of_screen_name` - number of numeric characters at the end of a user's screen name
- `is_protected` - binary feature of whether or not the account is protected
- `numbers_at_beginning_of_screen_name` - number of consecutive numeric characters at the beginning of a user's screen name
- `is_verified` - binary feature of whether or not the account is verified or not

We trained and compared five machine learning algorithms via the Machine Learning in R (mlr) package [22]. The five algorithms we chose were AdaBoost, Logistic Regression, Decision Trees, Random Forest and Naive Bayes.

To evaluate each of the algorithms, we measured their accuracy, precision, sensitivity, F1-score, area under ROC curve (AUROC), Cohen's kappa coefficient and Brier score. By using each of these metrics, we can assess how well each model identifies both bot and genuine accounts.

To compute the aforementioned metrics, we used 10-fold cross-validation. Cross-validation is a resampling procedure used to compare the performance of machine learning algorithms to each other or of those with different hyperparameters. Cross-validation helps assess how the model will perform on an unknown dataset and provide insight to if and how our model is overfitting the data.

In our case, we shuffled the data randomly and split it into 10 equal groups. We trained a model using nine of the slices of data and then evaluated it on the remaining slice. We repeated this process to evaluate each of the 10 groups. Each round we computed metrics, and then after the 10th round, we took the average of each metric.

Because we had two sets of bot accounts, we set up five tasks where we utilized the genuine accounts and a different combination or proportion of the social spam bots and crypto-giveaway bots. We set up these tasks to measure the effect of the sample size on the chosen algorithms and to evaluate how having a mixed set of bot accounts affected prediction. The full results from the 10-fold cross-validation for each task are in Section B of the Appendix.

In summary, the Random Forest classifier proved to perform the best irrespective of the bot data used for training. It consistently had an accuracy of approximately 98 percent and an AUC of .99. Also its Brier score was the lowest across each task.The Naive Bayes classifier proved to be the worst, as expected. It was the only model that didn't achieve at least a 90 percent accuracy.

System	Social Spam bots	Crypto-giveaway bots	Undersampled Crypto-giveaway bots	Social Spam bots + Crypto-giveaway bots	Undersampled Social Spam bots + Crypto-giveaway bots
AdaBoost Classifier	.9952	.9947	.9943	.9936	.9935
Logistic Regression	.9722	.9738	.9867	.9187	.9738
Decision Tree	.9781	.9706	.9739	.9740	.9728
Random Forest	.9948	.9946	.9942	.9938	.9939
Naive Bayes	.9424	.9449	.9486	.8927	.9248

AUC across each task

Using the metrics from cross-validation, we knew that the Random Forest model was the optimal choice; however, we wanted to assess how the model would perform if trained on one set of the bot data and then predicted on the other. We knew that each sample wasn't a representative sample of bots in the Twitter eco system, but we wanted to find out what we considered to be our test accuracy on known bots.

To do so, we trained three Random Forest models:

1. A dataset of the genuine accounts and the social spam bots
2. One which had the genuine accounts and a random sampling of 4,912 crypto-giveaway bots
3. Another that contained the genuine accounts and entire set of crypto-giveaway accounts.

We then used the three models to predict on the other class of bots.

In our case, the output of the Random Forest model is the probability that an account is a bot. Because we don't have any information in regards to the penalty or cost associated with either type of classification error, we use 0.5 as the threshold for whether an account is a bot or not.

With that threshold, we achieved an accuracy of 80 percent using the model trained with the social spam bots to predict on the crypto-giveaway bot accounts. The model trained on an equivalent number of crypto-giveaway accounts performed far worse by comparison with an accuracy of 49 percent on the social spam bots; however, when trained using the entire set of crypto-giveaway bots, we were able to achieve a 53 percent accuracy. One source of this minor increase in accuracy could be the increased amount of bot accounts trained on.

Training Data	Accounts Predicted on	Accuracy
Genuine + Social Spam Accounts	Crypto-giveaway accounts	80%
Genuine + 4912 Crypto-giveaway accounts	Social Spam Account	49%
Genuine + 7250 Crypto-giveaway accounts	Social Spam Account	53%

Performance of models when trained on other class of bots

Results on Unknown Accounts

Utilizing a dataset comprised of genuine accounts and all the bot accounts from each class of bots, we fit a Random Forest model once more. With this model, we then predicted the probability of whether or not 42 million accounts that we compiled utilizing the Twitter API were bots. To understand what our classifier finds important, we examined the differences between the accounts that had the highest probability to be a bot (top 20 percent) and the accounts with the lowest probability to be a bot (bottom 20 percent).

As we hoped, the bottom 20 percent contained the vast majority of the verified accounts in our dataset which is why the `ratio_followers_friends` is significantly higher. Verified accounts belong to users involved in media, sports, music, acting or other areas of interest [33]. Those accounts typically have a very high ratio of followers to friends. Upon further investigation of the accounts in the top 20 percent percent, our fitted model outputs a higher probability that older accounts with very little activity is a bot, rather than those with more activity.

Feature	Top 20%	Bottom 20%
ratio_followers_friends	0.022	27.48
screen_name_entropy	2.94	2.93
numbers_at_beginning_of_screen_name	0.17	0.037
tweet_rate	0.22	10.91
favorite_rate	0.19	6.60
numbers_at_end_of_screen_name	2.09	0.48

Classification via Tweet Behavior, Tweet Content and Account

Feature Selection

In the previous section, we only used 10 features based on account metadata to try to classify accounts. With such a quantity of features, the risk of overfitting (i.e., the chance that our model is fit too closely to our training data and may not generalize well when it encounters new data), is most likely smaller than the case in which we have 22 features (described in Section A of the appendix). Moreover, there is the potential that more of our features are highly correlated and/or don't provide us much additional signal given other features in the model. Generally, performing feature selection allows more interpretable models, shorter training times, and less data to keep around.

There are several methods to select a subset of features. Our method of choice was recursive feature elimination through the use of Random Forests. We implemented this using the caret [21] package. Put simply, this approach removes features recursively and builds a model on the features that remain. It uses the metric of importance (e.g., accuracy, root mean squared error, AUC, etc.) to identify which feature or subset of features contribute the most and ranks them.

Utilizing a Random Forest model to determine the best set of features, we found that on the Cresci dataset, we were able to achieve an AUC of 0.99 by only using `favorite_rate`, `is_default_profile`, `avg_reply_id_distance`, `avg_distinct_hours_tweeted` and `avg_num_users_mentioned_in_tweets`. However, our accuracy rate on the

crypto-giveaway accounts was only 83 percent. During feature selection, we are optimizing for the AUC on the training data; however, as discussed earlier, if the accounts we are predicting on behave differently, we will see much worse results.

Results on Unknown Data

The sample of data that we used in classifying accounts using the five features above was a sample of 692,906 accounts that had tweeted in the last 90 days. We did this to replicate the approach outlined before, which only used account metadata, but this time, we restricted our predictions to recently active accounts.

Using the five features from above, we found significant differences in the averages of each of the features when comparing the top 20 percentile of accounts deemed to be bots by our model and the bottom 20 percentile. Those accounts in the top 20 percentile were more likely to have a default profile, tweet over more hours and favorite fewer statuses.

Feature	Top 20 %	Bottom 20%
is_default_profile	.67%	60%
avg_num_users_mentioned_in_tweets	0.6738795	0.8812649
avg_distinct_hours_tweeted	5.4	4.9
favorite_rate	4.843299	22.852629
avg_reply_id_distance	1.87e17	9.6e16

Limitations, Considerations and Other Discussion

Because our dataset only consists of an account's last 200 tweets, we are only able to get a glimpse into their real behavior. It's possible for an account to exhibit behaviors of a normal user or behaviors that would only be possible through automation at different times.

As a result, our findings in the previous sections represent an interpretation of what the account looked and behaved like recently. If we considered an account's entire timeline, we could more accurately determine which accounts were behaving like bots. With that data, it is also possible to determine shifts in behavior as well.

An additional point of consideration is that our dataset of bot accounts consists of only two groups of bots -- one of which is made up of accounts from two years ago. Those sets of accounts are not representative of all bot accounts that may exist on the Twitter platform. Our results show that our accuracy significantly decreases when the algorithm sees a type of bot that is different in some way from the accounts included during training.

Another point that merits discussion is the cost of a false positive. We aren't privy to the support costs or any other effects that Twitter must take into account when trying to predict whether an account is a bot or not. Having a greater understanding of those costs would have an impact on the thresholds of what is or isn't a bot. Optimizing for cost would most likely affect the accuracy, precision and sensitivity for any method.

Crawling Botnets

After using these approaches to find initial sets of bots, the next steps are to determine if the bots are isolated or if they are part of a larger botnet controlled by a single entity.

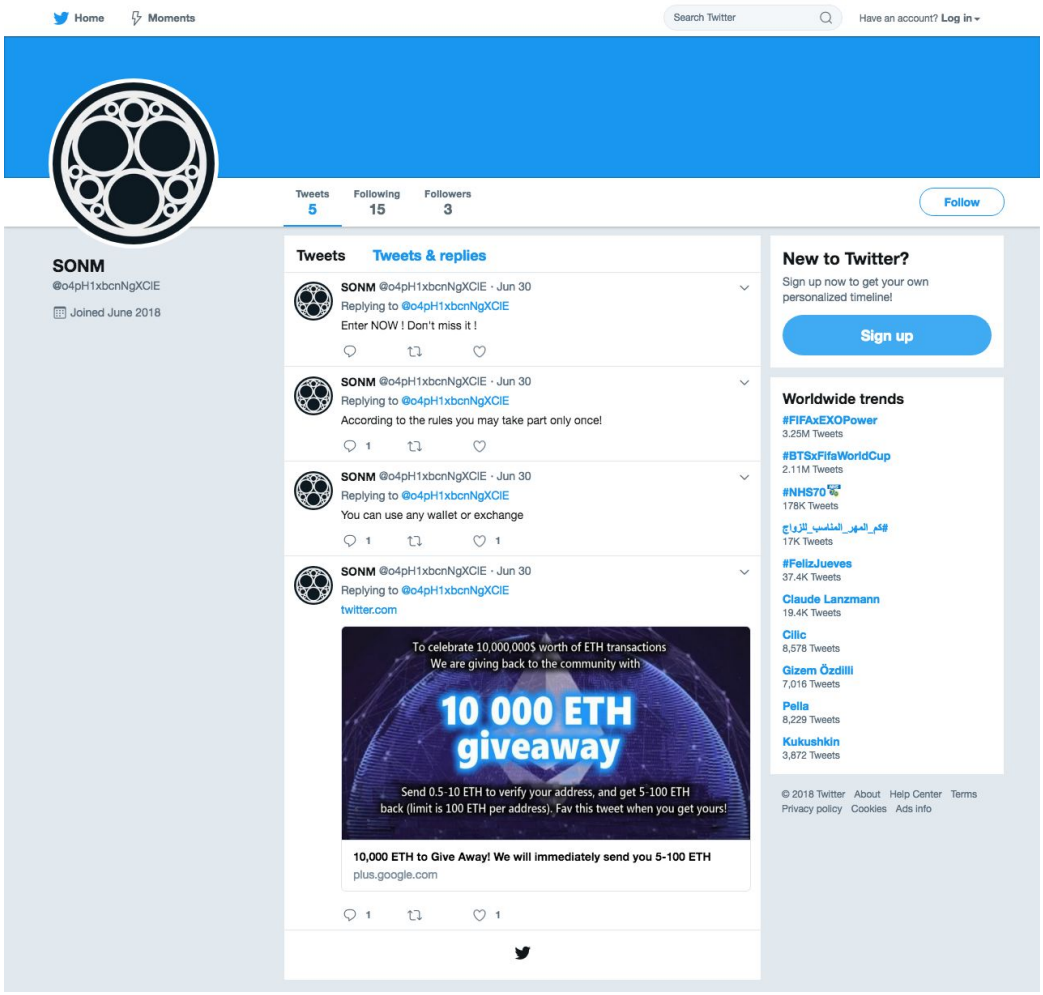
Botnets can be structured in different ways. For example, the Star Wars [5] and Bursty [6] botnets have a flat structure where each bot showed the same behavior. Alternatively, some botnets such as a diet-spam botnet discovered by Symantec [24] contain a more organized, multi-tiered structure with bots dedicated to different roles such as publishing spam, tweet amplification and fake followers.

The following section is a case study about a botnet we discovered that uses a multi-tiered structure similar to that of the diet-spam botnet to spread cryptocurrency scams.

Case Study: Cryptocurrency Scam Botnet

Initial Discovery

In late May 2018, we discovered what appeared to be automated accounts spoofing otherwise legitimate cryptocurrency Twitter accounts in an effort to spread a "giveaway" scam.



The typical operation of the bots involved first creating a spoofed account for a legitimate cryptocurrency-affiliated account. This spoofed account would have (what appeared to be) a randomly-generated screen name, and would copy the name and profile picture of the legitimate account.

To spread the spam, the bot would reply to a real tweet posted by the legitimate account. This reply would contain a link inviting the victim to take part in a cryptocurrency giveaway.

As discussed in the next “Evolution of the Botnet” section, this botnet used several methods to evade detection over the few months we were periodically monitoring it. However, early iterations of the botnet could be matched using basic signatures. We used these signatures with the `search/tweets` [25] API endpoint to gather the initial list of 7,250 bots to be used in exploratory data analysis and building our classifier.

We polled the same API endpoint on multiple occasions throughout June 2018 to determine if the botnet was still active, which resulted in a collection of 12,000 spam generating bots.

The initial set of bots provided an indication of what type of spam the botnet was publishing, but didn't reveal how the bots are organized and managed. To map out the structure behind the bots, we gathered a final small collection of bots to be used when performing a recursive crawl detailed in the "Mapping the Network" section below.

Evolution of the Botnet

Throughout the course of our research, we observed the accounts responsible for spreading the malicious links would use increasingly sophisticated techniques to avoid automated detection. Specifically, we observed the following techniques being used:

- Using unicode characters in tweets instead of traditional ASCII characters
- Adding various white spaces between words or punctuation
- Transitioning to spoofing celebrities and high-profile Twitter accounts in addition to cryptocurrency accounts
- Using screen names that are typos of a spoofed account's screen name
- Performing minor editing on the profile picture to avoid image detection

These attempts to thwart detection demonstrate the importance of analyzing an account holistically, including the metadata around the content. For example, these accounts will typically tweet in short bursts, causing the average time between tweets to be very low.

Mapping the Network

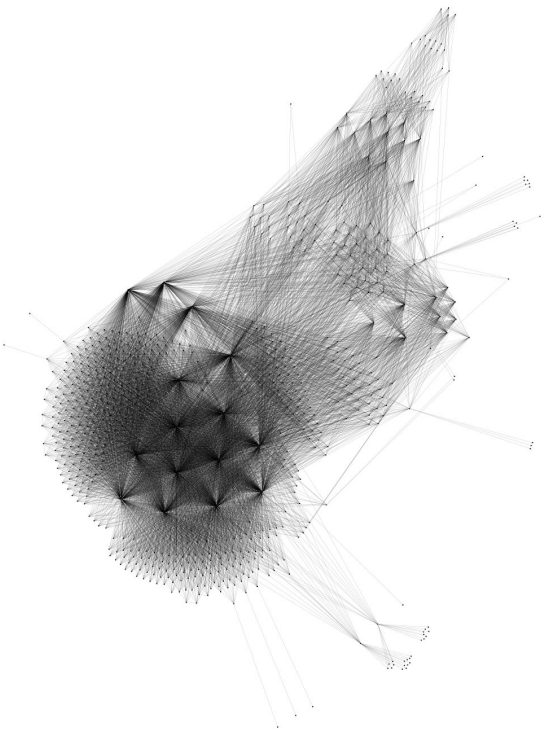
Mapping Followers

Once we determined how the bots operate, we began analyzing the structure of the botnet. We noticed that many of the bots followed what appeared to be the same accounts. We consider these accounts to be "hub accounts," since many bots follow them. It's unclear how these hub accounts directly contribute to the botnet operation, other than being a set of centralized accounts many of the bots follow. It's possible that these accounts aren't affiliated with the botnet and are randomly chosen accounts which the bots follow in an effort to appear legitimate.

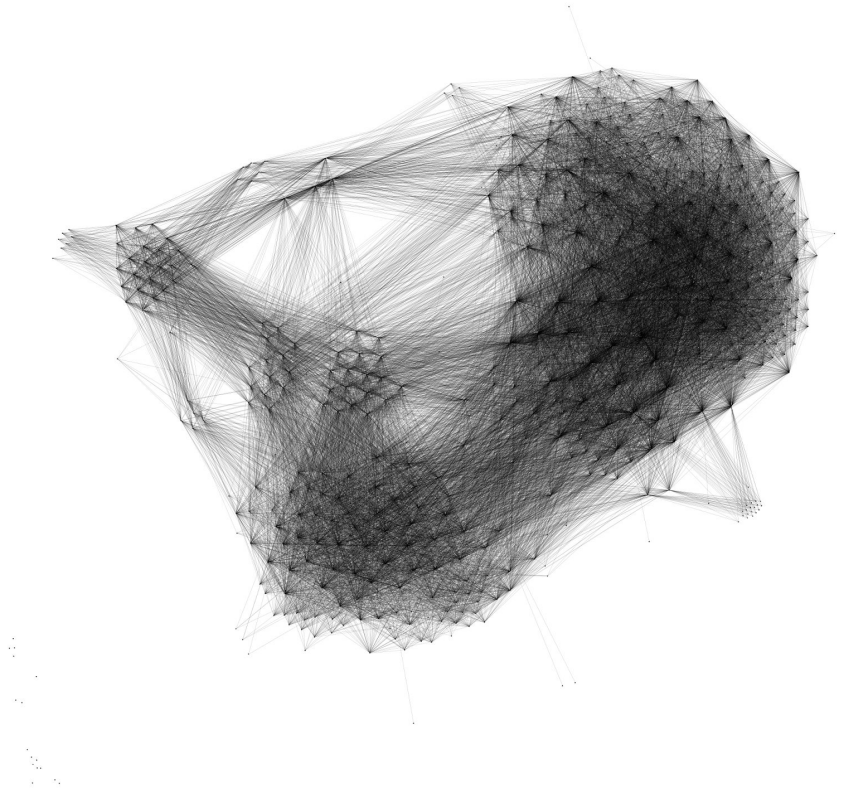
We then performed a recursive search, gathering which accounts each bot followed. Then, for each of these hub accounts, we gathered all their followers, checking to see if the followers were a cryptocurrency scam bot. If we determined it was a bot, we recursively searched the accounts that bot was following since these may be new hub accounts.

This allowed us to enumerate around 2,600 additional bots which were found to be spreading malicious links claiming to be cryptocurrency giveaways. Mapping these relationships revealed two notable loosely connected clusters.

The first cluster we discovered mostly follow hub accounts, creating a sparse community of bots all following the same accounts. The following graph maps these connections, with each hub account displayed as a highly-connected node:



The second cluster takes a different approach. The accounts in this cluster are highly connected in a mesh-like structure, where each bot follows other bots in the same community:



Mapping Amplification Bots

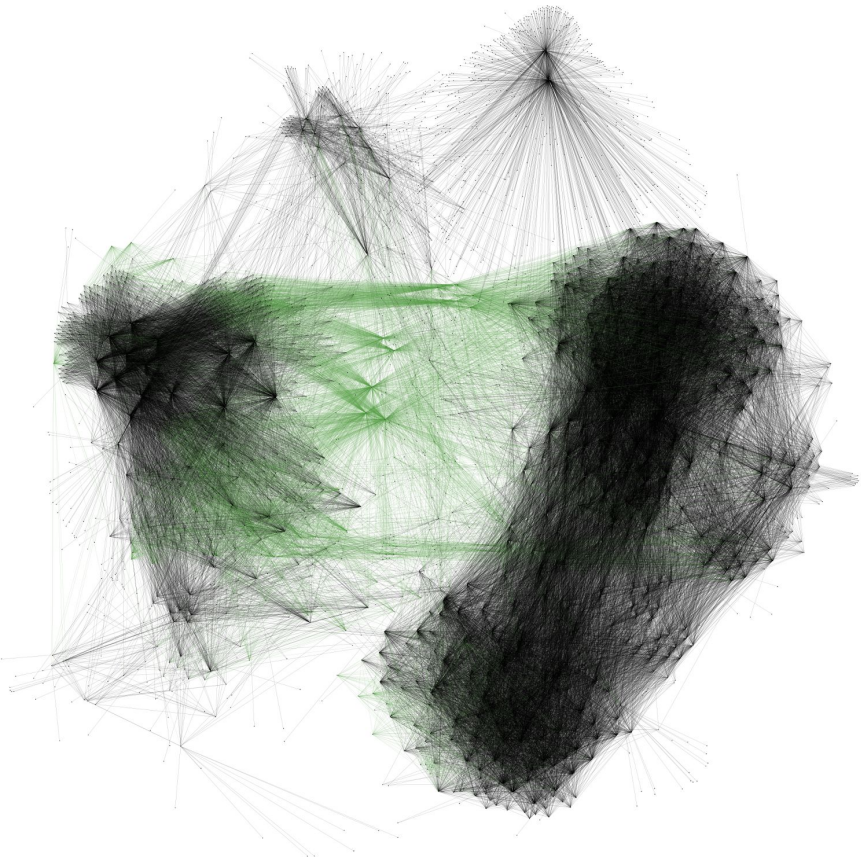
Amplification bots are accounts that exist to like tweets in order to artificially inflate the tweet's popularity. To make the cryptocurrency scam appear legitimate, we noticed that amplification bots were used to increase the number of likes for the tweets sent by bots.

The only activity these amplification bots had were liking the cryptocurrency scam tweets from a variety of bots:

We wanted to map out the relationships between amplification bots and the bots they support. To do this, for every tweet from the cryptocurrency scam bots, we determined which accounts liked the tweet (see the “Approaches that Didn’t Work” section for difficulties we encountered when performing this mapping). If the account had the same characteristics of an amplification bot, we used the `favorites/list` [26] endpoint to enumerate every status the amplification bot has liked.

Searching through the other statuses liked by amplification bots helped us discover previously unknown cryptocurrency scam bots. Each time we discovered a new scam bot spreading malicious links, we performed the same social network gathering as above, looking for even more connected bots.

This resulted in a three-tiered botnet structure consisting of the scam publishing bots, the hub accounts (if any) the bots were following, and the amplification bots that like each created tweet. The mapping shows that the amplification bots like tweets from both clusters, binding them together. This final mapping is shown below, where “following/follower” relationships are shown as black edges and “like” relationships are shown in green:



At a high level, this case study demonstrates that, after finding initial bots using the tools and techniques described in this paper, **a thread can be followed that can result in the unraveling of an entire botnet.**

While gathering social network connections can be expensive due to the rate limits in place at the time of this writing, gathering this information in a targeted fashion can be very effective in identifying new potential bot accounts after initial sets of bots have been gathered. This type of network analysis also helps reveal the structure of the botnet, helping provide characteristics that can be used to identify similar botnets in future work.

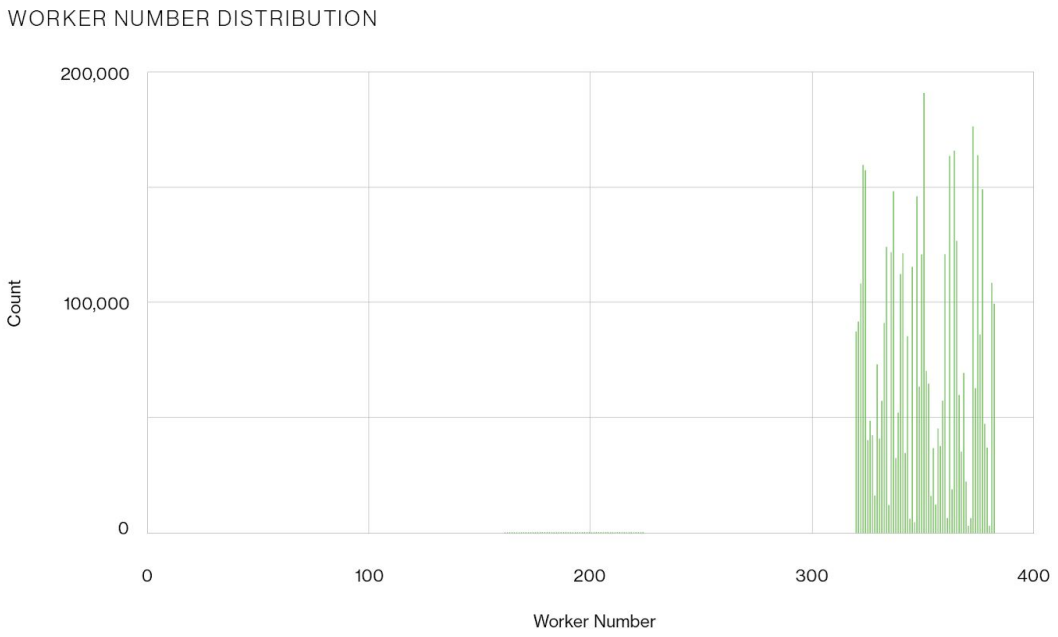
Approaches That Didn't Work

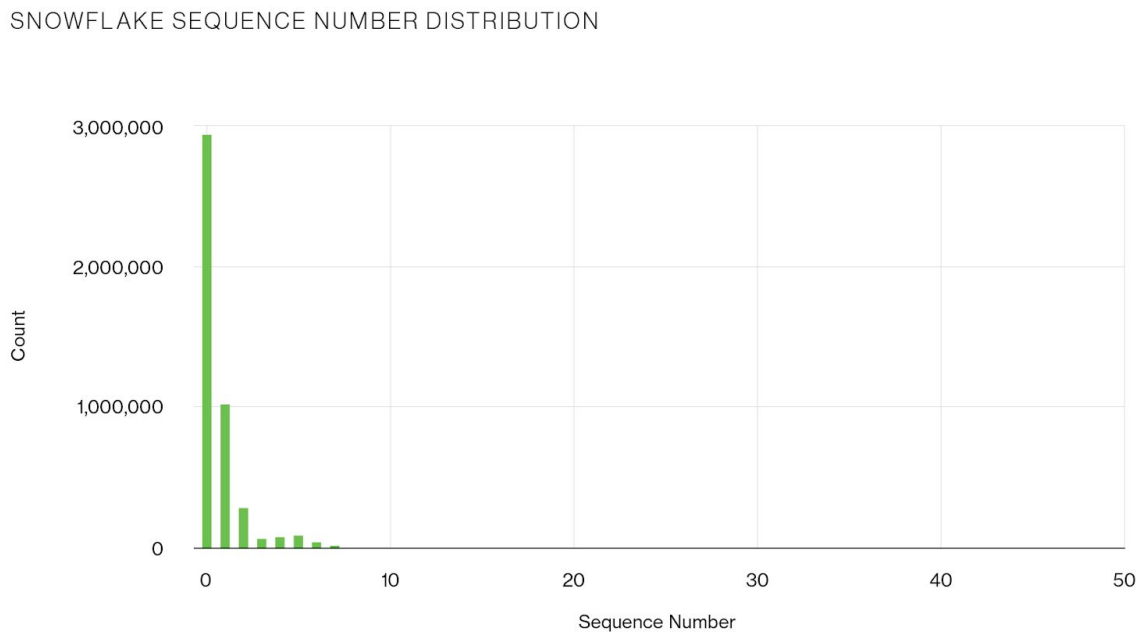
Snowflake ID Backtracking

The "How Twitter Assigns User IDs" section above describes the transition to 64-bit IDs generated by a system called Snowflake. These IDs are created from a timestamp, a worker number and a sequence number.

Our goal was to determine if we could use the IDs we previously gathered via streaming to generate valid Snowflake IDs that could then be sent to the `users/lookup` endpoint to find new accounts. The idea is that new accounts are likely created frequently, so for each ID, we could decrement the sequence number and create a new candidate ID that would be a possible account ID.

To do this, we split each ID into the different components and measured the frequency of values seen. The following charts show the distribution for worker numbers and sequence numbers.





These charts show that, while there are multiple bits reserved for these values, the actual range of values is fairly predictable. This suggests that even just predicting random Snowflake values may be possible by choosing a random timestamp in the valid date range, picking a worker number in the observed range, and starting with sequence number 0.

However, for our approach, we opted for an even simpler solution. Since the distribution of sequence numbers drops significantly, we can infer that the sequence number is reset for each new timestamp. This is further confirmed in the (now deprecated) Snowflake source code [27].

To generate our candidate IDs, we filtered our existing Snowflake ID dataset to those with a sequence number greater than 0. Then, we generated new IDs for each sequence number until we reached sequence number 0. Finally, we submitted the generated IDs to the `users/lookup` endpoint to potentially gather new accounts.

While this resulted in finding some accounts, the overall success rate was well below 1 percent. The results suggest that workers generating snowflake IDs are responsible for generating the IDs for many types of Twitter content in addition to just accounts.

Since we could assume that there will be many more tweets than accounts, we changed our approach to use the `statuses_lookup` endpoint instead of the `users/lookup` endpoint. This increased the success rate dramatically to approximately 13 percent, proving the method to be effective at finding users who have recently tweeted, since each tweet object contains the original user object.

However, our goal was to find accounts that were created around a particular date and, to that end, this approach was unsuccessful.

Finding Who Favorited a Tweet

As mentioned in the “Crawling Botnets” section, a common botnet structure includes amplification bots whose sole function is to like tweets containing malicious content in an attempt to make the content appear legitimate. It’s often the case that these accounts are the only links between other spam-creating bots, making them useful for researchers.

At the time of this writing, we were unable to find a published API endpoint for determining which accounts liked a particular tweet. This behavior currently only exists for discovering retweets of a particular status using the `statuses/retweeters/ids` endpoint [28].

For our case study, in order to determine which accounts liked a particular status, we had to rely on an unofficial API endpoint, which is not ideal when performing large-scale research. The solution could be to create a new API endpoint, e.g. `statuses/favorites/ids` or similar.

Next Steps

During the course of this research, which spanned May 2018 through July 2018, Twitter announced that they are taking more proactive action against both automated spam and malicious content [29], identifying and challenging “more than 9.9 million potentially spammy or automated accounts per week.” In a follow-up blog post, Twitter also described their plans to remove accounts that had been previously locked due to suspicious activity from follower counts [30].

We’re hopeful that these increased investments will be effective in combating spam and malicious content, however, we don’t consider the problem solved. The case study presented in this paper demonstrates that organized botnets are still active and can be discovered with straightforward analysis.

During this research and in our conversations with Twitter when sharing our analysis, an area that emerged as being important to any future research was the difference between the view of Twitter built through its API and what users see in the user interface (UI). According to the company, Twitter is actively working to hide malicious content from being visible in areas like search and conversations, though the content can still be visible via the API. Twitter cites that ‘less than 5%’ of accounts are spam related.

Differences between data exposed via the UI and API, and the security ramifications of these differences, is an area we are excited to explore further in future work.

By open-sourcing the tools and techniques we developed during this research, we hope to enable researchers to continue building on our work, creating new techniques to identify and flag malicious bots, and helping to keep Twitter and other social networks a place for healthy online discussion and community.

References

[1] TwitterDev. *“Docs - Twitter Developers.”* Retrieved from https://developer.twitter.com/en/docs
[2] TwitterDev. *“User Object - Twitter Developers.”* Retrieved from developer.twitter.com/en/docs/tweets/data-dictionary/overview/user-object
[3] TwitterDev. *“GET users/lookup - Twitter Developers.”* Retrieved from https://developer.twitter.com/en/docs/accounts-and-users/follow-search-get-users/api-reference/get-users-lookup
[4] Krishnamurthy, B., Gill, P., & Arlitt, M. (2008). A few chirps about twitter. Proceedings of the First Workshop on Online Social Networks - WOSP ’08, 19. http://doi.org/10.1145/1397735.1397741
[5] Echeverría, J., & Zhou, S. (2017). Discovery, Retrieval, and Analysis of “Star Wars” botnet in Twitter. http://doi.org/10.1145/3110025.3110074
[6] Echeverria, J., Besel, C., & Zhou, S. (2017). Discovery of the Twitter Bursty Botnet, 1–9. Retrieved from http://arxiv.org/abs/1709.06740
[7] Roomann-Kurrik, A. (2013, January 22). “*Moving to 64-bit Twitter User IDs.”* Retrieved from https://blog.twitter.com/developer/en_us/a/2013/64-bit-twitter-user-idpocalypse.html
[8] King, R. (2010, June 1). *“Announcing Snowflake.”* Retrieved from https://blog.twitter.com/engineering/en_us/a/2010/announcing-snowflake.html
[9] TwitterDev. *“GET statuses/sample - Twitter Developers.”* Retrieved from https://developer.twitter.com/en/docs/tweets/sample-realtime/overview/GET_statuse_sample
[10] TwitterDev. *“GET statuses/user_timeline - Twitter Developers.”* Retrieved from https://developer.twitter.com/en/docs/tweets/timelines/api-reference/get-statuses-user_timeline.html
[11] TwitterDev. *“GET statuses/lookup - Twitter Developers.”* Retrieved from https://developer.twitter.com/en/docs/tweets/post-and-engage/api-reference/get-statuses-lookup
[12] TwitterDev. *“GET friends/ids - Twitter Developers.”* Retrieved from https://developer.twitter.com/en/docs/accounts-and-users/follow-search-get-users/api-reference/get-friends-ids
[13] TwitterDev. *“GET followers/ids - Twitter Developers.”* Retrieved from https://developer.twitter.com/en/docs/accounts-and-users/follow-search-get-users/api-reference/get-followers-ids
[14] Gephi. (2017). *“Gephi - The Open Graph Viz Platform.”* Retrieved from https://gephi.org/
[15] Neo4j. *“The Neo4j Graph Platform – The #1 Platform for Connected Data.”* Retrieved from https://neo4j.com/
[16] Maksym Gabielkov, Arnaud Legout. The Complete Picture of the Twitter Social Graph. ACM
CoNEXT 2012 Student Workshop, Dec 2012, Nice, France. 2012. <hal-00752934v3>

[17] Confessore, Nicholas, et al. “The Follower Factory.” *The New York Times*, The New York Times, 27 Jan. 2018, www.nytimes.com/interactive/2018/01/27/technology/social-media-bots.html.
[18] Subrahmanian, V. S., Azaria, A., Durst, S., Vadim, K., Galstyan, A., Lerman, K., … Menczer, F. (n.d.). The DARPA TWITTER BOT CHALLENGE. Retrieved from https://arxiv.org/pdf/1601.05140.pdf
[19] Kudugunta, S., & Ferrara, E. (2018). Deep Neural Networks for Bot Detection. Retrieved from http://arxiv.org/abs/1802.04289
[20] Cresci, S., Di Pietro, R., Petrocchi, M., Spognardi, A., & Tesconi, M. (2017). The paradigm-shift of social spambots: Evidence, theories, and tools for the arms race. Proceedings of the 26th International Conference on World Wide Web Companion - WWW ’17 Companion, 963–972. http://doi.org/10.1145/3041021.3055135
[21] Cresci, S., Di Pietro, R., Petrocchi, M., Spognardi, A., & Tesconi, M. (2016). DNA-inspired online behavioral modeling and its application to spambot detection. IEEE Intelligent Systems, 31(5), 58-64.
[22] Bischl B, Lang M, Kotthoff L, Schiffner J, Richter J, Studerus E, Casalicchio G and Jones Z (2016). “mlr: Machine Learning in R.” Journal of Machine Learning Research, 17(170), pp. 1-5. http://jmlr.org/papers/v17/15-066.html.
[23] Kuhn, M. (2008). Caret package. Journal of Statistical Software, 28(5)
[24] Narang, S. (2015). Uncovering a persistent diet spam operation on Twitter, 21. Retrieved from http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/uncovering-a-persistent-diet-spam-operation-on-twitter.pdf
[25] TwitterDev. *“Standard search API - Twitter Developers.”* Retrieved from https://developer.twitter.com/en/docs/tweets/search/api-reference/get-search-tweets
[26] TwitterDev. *“GET favorites/list - Twitter Developers.”* Retrieved from https://developer.twitter.com/en/docs/tweets/post-and-engage/api-reference/get-favorites-list
[27] King, R. (2012, March 1). “*twitter/snowflake”*. Retrieved from https://github.com/twitter/snowflake/blob/b3f6a3c6ca8e1b6847baa6ff42bf72201e2c2231/src/main/scala/com/twitter/service/snowflake/IdWorker.scala#L81
[28] TwitterDev. *“GET statuses/retweeters/ids - Twitter Developers.”* Retrieved from https://developer.twitter.com/en/docs/tweets/post-and-engage/api-reference/get-statuses-retweeters-ids
[29] Roth, Y., Harvey, D. (2018, June 26). *“How Twitter is fighting spam and malicious automation”*. Retrieved from https://blog.twitter.com/official/en_us/topics/company/2018/how-twitter-is-fighting-spam-and-malicious-automation.html
[30] Gadde, V. (2018, July 11). *“Confidence in follower counts”*. Retrieved from https://blog.twitter.com/official/en_us/topics/company/2018/Confidence-in-Follower-Counts.html
[31] TwitterDev. *“Twitter IDs (snowflake) - Twitter Developers.”* Retrieved from https://developer.twitter.com/en/docs/basics/twitter-ids.html

[32] Thomas, K., McCoy, D., Grier, C., Kolcz, A., & Paxson, V. (2013). Trafficking Fraudulent Accounts: The Role of the Underground Market in Twitter Spam and Abuse. *USENIX Security Symposium*, 195–210.
[33] About verified accounts. (n.d.). Retrieved from https://help.twitter.com/en/managing-your-account/about-twitter-verified-accounts
[34] TwitterAudit. "*Twitter Audit | How many of your followers are real?"*. Retrieved from https://www.twitteraudit.com/
[35] Duo Labs. "*duo-labs/twitterbots"* Retrieved from https://github.com/duo-labs/twitterbots
[36] Varol, O., Ferrara, E., Davis, C. A., Menczer, F., & Flammini, A. (2017). Online Human-Bot Interactions: Detection, Estimation, and Characterization. Retrieved from http://arxiv.org/abs/1703.03107
[37] Davis, C. A., Varol, O., Ferrara, E., Flammini, A., & Menczer, F. (2016, April). Botornot: A system to evaluate social bots. In Proceedings of the 25th International Conference Companion on World Wide Web (pp. 273-274). International World Wide Web Conferences Steering Committee

Appendix

A. Feature List for Machine Learning Algorithms

Because of the fidelity in the Cresci dataset, we were only able to include a subset of the attributes we computed when trying to identify heuristics.

Feature	**Description**
`favorite_rate`	Ratio of favorites to account age
`ratio_followers_friends`	The number of followers divided by the number of friends or accounts the user is following
`tweet_rate`	Ratio of number of tweets to account age
`screen_name_entropy`	Shannon entropy of a user's screen name
`numbers_at_end_of_screen_name`	Number of numeric characters at the end of a user's screen name
`numbers_at_beginning_of_screen_name`	Number of consecutive numeric characters at the beginning of a user's screen name
`is_protected`	Binary feature of whether or not the account is protected
`is_verified`	Binary feature of whether or not the account is verified
`is_geo_enabled`	Binary feature of whether or not geo coordinates are accessible
`is_default_profile`	Binary feature of whether or not the user has altered the default profile
`percent_retweets`	Percent of tweets that were retweets
`percent_tweets_have_urls`	Percent of tweets that contain URLs
`avg_num_urls_in_tweets`	Average number of URLs in tweets
`number_distinct_sources`	Number of distinct sources tweeted from

percent_tweets_have_hastags	Percentage of tweets that contain hashtags
avg_num_users_mentioned	Average number of hashtags in tweets
percent_tweets_have_users_mentioned	Percentage of tweets that mention users
avg_reply_distance	Average distance between user IDs of messages replied
duplicates	Average number of times a user tweets the same content more than once per day
avg_time_between_tweets	The average amount of time between tweets
num_users_replied	The number of users replied to
avg_distinct_hours_tweeted	The average number of distinct hours tweeted in a day

B. Cross-Validation Results

Positive Sample: Social Spam Bots

System	Accuracy	Precision	Recall	F1-Score	AUROC	Kappa	Brier
AdaBoost Classifier	.9833	.9889	.9825	.9857	.9952	.9655	.0142
Logistic Regression	.9257	.9008	.9808	.9391	.9722	.8439	.0567
Decision Tree	.9775	.9872	.9741	.9807	.9781	.9535	.0233
Random Forest	.9843	.9906	.9825	.9865	.9948	.9675	.0135
Naive Bayes	.8357	.7934	.9721	.8736	.9424	.6446	.1283

Positive Sample: Crypto-Giveaway Bots

System	Accuracy	Precision	Recall	F1-Score	AUROC	Kappa	Brier

AdaBoost Classifier	0.9742	.9897	.9718	.9807	.9947	.9417	.0216
Logistic Regression	.9413	.9414	.9739	.9573	.9738	.8635	.0480
Decision Tree	.9662	.9829	.9669	.9748	.9706	.9236	.0319
Random Forest	.9745	.9905	.9714	.9809	.9946	.9424	.0205
Naive Bayes	.8770	.8550	.9851	.9154	.9449	.6939	.0951

Positive Sample: Undersampled Crypto-Giveaway Bots

System	Accuracy	Precision	Recall	F1-Score	AUROC	Kappa	Brier
AdaBoost Classifier	.9752	.9884	.9627	.9753	.9943	.9504	.0194
Logistic Regression	.9451	.9254	.9704	.9473	.9867	.8899	.0517
Decision Tree	.9697	.9752	.9652	.9702	.9739	.9394	.0273
Random Forest	.9770	.9895	.9652	.9772	.9942	.9540	.0184
Naive Bayes	.8448	.7724	.9864	.8663	.9486	.6875	.10465

Positive Sample: Social Spam and Crypto-Giveaway Bots

System	Accuracy	Precision	Recall	F1-Score	AUROC	Kappa	Brier
AdaBoost Classifier	.9736	.9895	.9764	.9823	.9936	.9247	.0212
Logistic	.9212	.9178	.9872	.9512	.9187	.7480	.0654

Regression							
Decision Tree	.9698	.9913	.9697	.99804	.9740	.9149	.0277
Random Forest	.9750	.9913	.9765	.9838	.9938	.9288	.0202
Naive Bayes	.9180	.9180	.9823	.9490	.8927	.7394	.0723

Positive Sample: Undersampled Social Spam and Crypto-Giveaway Bots

System	Accuracy	Precision	Recall	F1-Score	AUC/ROC	Kappa	Brier
AdaBoost Classifier	.9758	.9866	.9681	.9772	.9935	.9514	.0210
Logistic Regression	.9030	.8844	.9423	.9123	.9738	.8040	.0743
Decision Tree	.9710	.9823	.9629	.9727	.9728	.9418	.0266
Random Forest	.9782	.9886	.9706	.9795	.9939	.9562	.0193
Naive Bayes	.7942	.7275	.9856	.8368	.9248	.5746	.1777

Olabode Anise
@JustSayO
Olabode is a Data Scientist at Duo Security where he wrangles data, prototypes data-related features, and makes pretty graphs to support engineering, product management, and marketing efforts. Prior to Duo, Olabode studied usable security at the University of Florida. When he's not at work, he spends his time exploring data involved topics such as sports analytics, relative wages and cost of living across the United States.

Jordan Wright
@jw_sec
Jordan Wright is Principal R&D Engineer at Duo Security as a part of the Duo Labs team. He has experience on both the offensive and defensive side of infosec. He enjoys contributing to open-source software and performing security research.